Implementing field extensions of the form $\mathbb{Q}[\sqrt{b}]^*$

René Thiemann

February 23, 2021

Abstract

We apply data refinement to implement the real numbers, where we support all numbers in the field extension $\mathbb{Q}[\sqrt{b}]$, i.e., all numbers of the form $p + q\sqrt{b}$ for rational numbers $p$ and $q$ and some fixed natural number $b$. To this end, we also developed algorithms to precisely compute roots of a rational number, and to perform a factorization of natural numbers which eliminates duplicate prime factors.

Our results have been used to certify termination proofs which involve polynomial interpretations over the reals.

Contents

1 Introduction 1

2 Auxiliary lemmas which might be moved into the Isabelle distribution. 2

3 Prime products 2

4 A representation of real numbers via triples 9

5 A unique representation of real numbers via triples 19

1 Introduction

It has been shown that polynomial interpretations over the reals are strictly more powerful for termination proving than polynomial interpretations over the rationals. To this end, also automated termination prover started to generate such interpretations. [3, 4, 5, 7, 8]. However, for all current implementations, only reals of the form $p + q \cdot \sqrt{b}$ are generated where $b$ is some fixed natural number and $p$ and $q$ may be arbitrary rationals, i.e., we get numbers within $\mathbb{Q}[\sqrt{b}]$.

*This research is supported by FWF (Austrian Science Fund) project P22767-N13.
To support these termination proofs in our certifier CeTA [6], we therefore required executable functions on \( \mathbb{Q}[\sqrt{b}] \), which can then be used as an implementation type for the reals. Here, we used ideas from [1, 2] to provide a sufficiently powerful partial implementations via data refinement.

2 Auxiliary lemmas which might be moved into the Isabelle distribution.

```plaintext
theory Real-Impl-Auxiliary
imports
  HOL—Computational-Algebra.Primes
begin

lemma multiplicity-prime:
  assumes p: prime (i :: nat) and ji: j ≠ i
  shows multiplicity j i = 0
  using assms
  by (metis dvd-refl prime-nat-iff multiplicity-eq-zero-iff
  multiplicity-unit-left multiplicity-zero)

end
```

3 Prime products

```plaintext
theory Prime-Product
imports
  Real-Impl-Auxiliary
  Sqrt-Babylonian.Sqrt-Babylonian
begin

Prime products are natural numbers where no prime factor occurs more than once.

definition prime-product
  where prime-product (n :: nat) = (∀ p. prime p → multiplicity p n ≤ 1)

  The main property is that whenever \( b_1 \) and \( b_2 \) are different prime products, then \( p_1 + q_1 \sqrt{b_1} = p_2 + q_2 \sqrt{b_2} \) implies \( (p_1, q_1, b_1) = (p_2, q_2, b_2) \) for all rational numbers \( p_1, q_1, p_2, q_2 \). This is the key property to uniquely represent numbers in \( \mathbb{Q}[\sqrt{b}] \) by triples. In the following we develop an algorithm to decompose any natural number \( n \) into \( n = s^2 \cdot p \) for some \( s \) and prime product \( p \).

function prime-product-factor-main :: nat ⇒ nat ⇒ nat ⇒ nat ⇒ nat ⇒ nat × nat
  where
  prime-product-factor-main factor-sq factor-pr limit n i =
  (if i ≤ limit ∧ i ≥ 2 then
  (if i dvd n
  ```
then \((\text{let } n' = n \div i \text{ in})\)
\((\text{if } i \text{ dvd } n' \text{ then})\)
\((\text{let } n'' = n' \div i \text{ in})\)
\((\text{prime-product-factor-main } (\text{factor-sq} \ast i) \text{ factor-pr } (\text{nat } (\text{root-nat-floor} \ 3 \ n'')) \ n'' \ i)\)
\((\text{else})\)
\((\text{(case sqrt-nat } n' \text{ of})\)
\((\text{Cons sn - } \Rightarrow (\text{factor-sq} \ast \text{sn} \ast \text{factor-pr} \ast i)\)
\((\text{| [] } \Rightarrow \text{prime-product-factor-main } \text{factor-sq} (\text{factor-pr} \ast i) (\text{nat } (\text{root-nat-floor} \ 3 \ n')) \ n' \ (\text{Suc } i)\)
\))
\)
\)
\(\text{else}\)
\(\text{(prime-product-factor-main } \text{factor-sq} \text{ factor-pr } \text{limit } n \ (\text{Suc } i)\)
\)
\(\text{else}\)
\((\text{factor-sq}, \text{factor-pr} \ast n)\) by pat-completeness auto

**termination**

**proof** –

\(\text{let } ?m1 = \lambda (\text{factor-sq} :: \text{nat}, \text{factor-pr} :: \text{nat}, \text{limit} :: \text{nat}, n :: \text{nat}, i :: \text{nat}). \ n\)
\(\text{let } ?m2 = \lambda (\text{factor-sq}, \text{factor-pr}, \text{limit}, n, i). \ (\text{Suc limit} - i)\)
\{\text{fix } i\}
\(\text{have } 2 \leq i \Rightarrow \text{Suc } 0 < i \ast i \ \text{using one-less-mult[of } i \text{ by auto}\)
\}\text{note } * = \text{this}\)
\(\text{show } ?\text{thesis}\)
\(\text{using } \text{wf-measures [of } ?m1, ?m2]\)
\(\text{by rule (auto simp add: } * \text{ elim!: dvdE split: if-splits)\)
\qed

**lemma prime-product-factor-main**: assumes \((\exists \ s. \ s \ast s = n)\)
\(\text{and limit} = \text{nat } (\text{root-nat-floor} \ 3 \ n)\)
\(\text{and } m = \text{factor-sq} \ast \text{factor-sq} \ast \text{factor-pr} \ast n\)
\(\text{and prime-product-factor-main } \text{factor-sq} \text{ factor-pr } \text{factor-pr} \ast \text{factor-pr} \text{ limit } n \ i = (\text{sq}, p)\)
\(\text{and } i \geq 2\)
\(\text{and } \bigwedge \ j. \ j \geq 2 \Rightarrow j < i \Rightarrow \neg j \text{ dvd } n\)
\(\text{and } \bigwedge \ j. \ \text{prime } j \Rightarrow j < i \Rightarrow \text{multiplicity } j \text{ factor-pr} \leq 1\)
\(\text{and } \bigwedge \ j. \ \text{prime } j \Rightarrow j \geq i \Rightarrow \text{multiplicity } j \text{ factor-pr} = 0\)
\(\text{and } \text{factor-pr} > 0\)
\(\text{shows } m = \text{sq} * \text{sq} * p \land \text{prime-product } p\)
\(\text{using } \text{assms}\)

**proof** (induct factor-sq factor-pr \text{factor-pr} \text{factor-pr} \text{limit} \text{n} \ \text{i} rule: prime-product-factor-main.induct)
\(\text{case } (1 \text{ factor-sq factor-pr limit } n \ i)\)
\(\text{note } IH = \text{I}(1-3)\)
\(\text{note premis} = \text{I}(4-\)
\(\text{note simp = premis[4]}[\text{unfolded prime-product-factor-main.simps[of factor-sq factor-pr limit n i]]}\)
\(\text{show } ?\text{case}\)
proof (cases \(\mathit{i} \leq \mathit{limit}\))

  case \(\mathit{True}\)

    note \(\mathit{i} = \mathit{this}\)

    with \(\text{prems}(5)\) have \(\mathit{cond} : \mathit{i} \leq \mathit{limit} \land \mathit{i} \geq 2\) \(\Rightarrow (\mathit{i} \leq \mathit{limit} \land \mathit{i} \geq 2) = \mathit{True}\) by blast+

    note \(\mathit{IH} = \mathit{IH}[(\mathit{OF} \ \mathit{cond})]

    note \(\mathit{simp} = \mathit{simp}[\text{unfolded} \ast \text{if-True}]\)

    show \(?\mathit{thesis}\)

  proof (cases \(\mathit{i} \vdash \mathit{n}\))

    case \(\mathit{False}\)

    hence \(*\) : \((\mathit{i} \vdash \mathit{n}) = \mathit{False}\) by simp

    note \(\mathit{simp} = \mathit{simp}[\text{unfolded} \ast \text{if-False}]\)

    note \(\mathit{IH} = \mathit{IH}(3) [(\mathit{OF} \ False \ \text{prems}(1-3) \ \text{simp}]\)

    show \(?\mathit{thesis}\)

  proof (rule \(\mathit{IH}\))

    fix \(\mathit{j}\)

    assume \(2: 2 \leq j \wedge j < \mathit{Suc} \ \mathit{i}\)

    from \(\text{prems}(6) [(\mathit{OF} \ 2)] \ \mathit{j} \ \mathit{False}\)

    show \(\neg \mathit{j} \vdash \mathit{n} \ \mathit{by} \ (\mathit{cases} \ \mathit{j} = \mathit{i}, \ \text{auto})\)

  next

    fix \(\mathit{j}\) :: nat

    assume \(j: j < \mathit{Suc} \ \mathit{i} \ \mathit{prime} \ \mathit{j}\)

    with \(\text{prems}(7-8) [\text{of} \ \mathit{j}]\)

    show \(\mathit{multiplicity} \ \mathit{j} \ \mathit{factor-pr} \leq 1\) by \(\mathit{cases} \ \mathit{j} = \mathit{i}, \ \text{auto}\)

  qed \(\mathit{(insert} \ \text{prems}(8-9) \ \text{cond, auto})\)

next

  case \(\mathit{True}\)

    note \(\mathit{mod} = \mathit{this}\)

    hence \(*\) : \((\mathit{i} \vdash \mathit{n}) = \mathit{True}\) by simp

    note \(\mathit{simp} = \mathit{simp}[\text{unfolded} \ast \text{if-True} \ \text{Let-def}]\)

    note \(\mathit{IH} = \mathit{IH}(1,2) [(\mathit{OF} \ \mathit{True} \ \text{refl}]\)

    show \(?\mathit{thesis}\)

  proof (rule \(\mathit{IH}\))

    show \(\mathit{m} = \mathit{factor-sq} \ast \mathit{i} \ast (\mathit{factor-sq} \ast \mathit{i}) \ast \mathit{factor-pr} \ast (\mathit{n} \ \mathit{div} \ \mathit{i} \ \mathit{div} \ \mathit{i})\)

      unfolding \(\text{prems}(3) \ \text{n'-def}[\text{symmetric}]\)

      unfolding \(\text{n by (auto simp: field-simps)}\)

    next

      fix \(\mathit{j}\)

      assume \(2: \mathit{2} \leq j \wedge j < \mathit{i}\)

      from \(\text{prems}(6) [(\mathit{OF} \ \text{this}) \ \mathit{have} \ \neg \mathit{j} \ \mathit{div} \ \mathit{n} \ \mathit{by} \ \text{auto}\)

      thus \(\neg \mathit{j} \ \mathit{div} \ \mathit{n} \ \mathit{div} \ \mathit{i} \ \mathit{div} \ \mathit{i}\)

      by \(\text{metis dvd-mult} \ \mathit{n} \ \text{n'-def [mult.commute]}\)

    next

  qed
show \( \neg (\exists \ s. \ s \ast s = n \div i \div i) \)
proof
assume \( \exists \ s. \ s \ast s = n \div i \div i \)
then obtain \( s \) where \( s \ast s = n \div i \div i \) by auto
hence \( (s \ast i) \ast (s \ast i) = n \) unfolding \( n \) by auto
with prems(1) show False by blast
qed
qed

next

case False
define \( n' \) where \( n' = n \div i \)
from mod True have \( n = n' \ast i \) by (auto simp: n'-def dvd-eq-mod-eq-0)
have prime: prime \( i \)
  unfolding prime-nat-iff
proof (intro conjI allI impI)
fix \( m \)
assume \( m: m \divd n \) unfolding \( n \) by auto
with prems(6)[of \( m \)] have choice: \( m \leq 1 \lor m \geq i \) by arith
from \( m \) prems(5) have \( m > 0 \)
  by (metis dvd-0-left-iff le0 le-antisym neg0-conv zero-neq-numeral)
with choice have choice: \( m = 1 \lor m \geq i \) by arith
from \( m \) prems(5) have \( m \leq i \)
  by (metis False div-by-0 dvd-refl dvd-imp-le gr0I)
with choice show \( m = 1 \lor m = i \) by auto
qed (insert prems(5), auto)
from False have \( (i \divd n \div i) = False \) by auto
note simp = simp[unfolded this if-False]
note IH = IH(2)[OF False - - refl]
from prime have \( i > 0 \) by (simp add: prime-gt-0-nat)

show \(?thesis \)
proof (cases sqrt-nat \( (n \div i) \))
case (Cons \( s \))
  note simp = simp[unfolded Cons list.simps]
  hence sq: \( sq = \text{factor-sq} \ast s \) and \( p: p = \text{factor-pr} \ast i \) by auto
  from arg-cong[OF Cons, of set] have \( s \ast s = n \div i \) by auto
  have pp: prime-product \( \text{factor-pr} \ast i \)
    unfolding prime-product-def
  proof safe
  fix \( m :: \text{nat} \) assume \( m: \text{prime} \ m \)
  consider \( i < m \ | \ i > m \ | \ i = m \) by force
  thus multiplicity \( m \) \( \text{factor-pr} \ast i \) \( \leq 1 \)
  by cases (insert prems(7)[of \( m \)] prems(8)[of \( m \)] prems(9) \( \ast i > 0 \): prime \( m \))
  simp-all add: multiplicity-prime prime-elem-multiplicity-mult-distrib)
qed
show \(?thesis \) unfolding sq \( p \) prems(3) \( n \) unfolding n'-def \( s[\text{symmetric}] \)
using \textit{pp} by \textit{auto}

next
case \textit{Nil}
\textbf{note simp = simp[unfolded \textit{Nil} list.simps]}
from \textit{arg-cong[OF \textit{Nil}, of set]} \textbf{have }\neg \left(\exists \ x. \ x \ast x = n \text{ div } i\right) \textbf{ by simp}
\textbf{note IH = IH[OF \textit{Nil} this - simp]}
\textbf{show }?\textit{thesis}
\textbf{proof }\left(\textit{rule }\textbf{IH}\right)
\textbf{show }m = \text{factor-sq} \ast \text{factor-sq} \ast \left(\text{factor-pr} \ast i\right) \ast \left(n \text{ div } i\right)
  \textbf{unfolding }\text{prems}(\text{3})\left[n\right] \textbf{by auto}
next
\textbf{fix }j
\textbf{assume }*: 2 \leq j j < \textit{Suc }i
\textbf{show }\neg \text{ j dvd } n \text{ div } i
\textbf{proof}
  \textbf{assume }j: \text{ j dvd } n \text{ div } i
  \textbf{with }\text{False} \textbf{ have }j \neq i \textbf{ by auto}
  \textbf{with }* \textbf{ have }2 \leq j j < i \textbf{ by auto}
  \textbf{from prems}(\text{6})[OF this] \text{ j}
  \textbf{show }\text{False} \textbf{ unfolding }n
    \textbf{by }\left(\text{metis dvd-mult }n \text{-def mult.commute}\right)
\textbf{qed}

next
\textbf{fix }j :: \textit{nat}
\textbf{assume }\text{Suc }i \leq j \text{ and } j\text{-prime: prime }j
\textbf{hence }ij: i \leq j \text{ and } j \neq i \textbf{ by auto}
\textbf{have }0: \text{multiplicity }j \text{ i }= 0 \textbf{ using }j \textbf{ by }\left(\text{rule }\text{multiplicity-prime}\right)
\textbf{show }\text{multiplicity }j \left(\text{factor-pr} \ast i\right) = 0
  \textbf{unfolding prems}(\text{8})[OF j\text{-prime }ij] \text{ 0}
  \textbf{using }\text{prime }j\text{-prime }j \left(0 < \text{factor-pr}\right) \left(\text{multiplicity }j \text{ factor-pr }= 0\right)
    \textbf{by }\left(\text{subst prime-elem-multiplicity-mult-distrib}\right) \left(\text{auto simp: multiplicity-prime}\right)

next
\textbf{fix }j
\textbf{assume }j < \text{Suc }i \text{ and } j\text{-prime: prime }j
\textbf{hence }j < i \vee j = i \textbf{ by auto}
\textbf{thus }\text{multiplicity }j \left(\text{factor-pr} \ast i\right) \leq 1
\textbf{proof}
  \textbf{assume }j = i
  \textbf{with prems}(\text{8})[of i] \text{ prime }j\text{-prime }0 < \text{factor-pr} \textbf{ show }?\textit{thesis}
    \textbf{by }\left(\text{subst prime-elem-multiplicity-mult-distrib}\right) \text{auto}
next
\textbf{assume }ji: j < i
\textbf{hence }j \neq i \textbf{ by auto}
\textbf{from prems}(\text{7})[OF j\text{-prime }ji] \text{multiplicity-prime}[OF prime this]
  \text{prime }j\text{-prime }0 < \text{factor-pr} \textbf{ show }?\textit{thesis} \textbf{by }\left(\text{subst prime-elem-multiplicity-mult-distrib}\right) \text{auto}
\textbf{qed}
\textbf{qed }\left(\text{insert prems}(\text{5}, \text{9}), \text{auto}\right)
qed
qed
qed
next
case False
hence \((i \leq \text{limit} \land i \geq 2) = \text{False}\) by auto
note simp = simp[unfolded this if-False]
hence sq: \(sq = \text{factor-sq}\) and \(p: p = \text{factor-pr} \ast n\) by auto
show \(\text{thesis}\)
proof
show \(m = sq \ast sq \ast p\) unfolding sq p prems(3) by simp
show prime-product p unfolding prime-product-def
proof safe
fix \(m::\text{nat}\) assume \(m: \text{prime}\ m\)
from prems(1) have \(n1: n > 1\) by (cases n, auto, case-tac nat, auto)
then obtain \(k\) where \(n1: n > 0\) by auto
have \(i > \text{limit}\) using False by auto
from this[unfolded prems(2)] have less: \(\text{int} i \geq \text{root-nat-floor } 3 \ast n + 1\) by auto
have \(\text{int} n < (\text{root-nat-floor } 3 \ast n + 1) \sim 3\) by (rule root-nat-floor-upper, auto)
also have \(\ldots \leq \text{int} i \sim 3\) by (rule power-mono[OF less, of 3], auto)
finally have \(\text{nat} i: \text{int} i < \text{root-nat-floor upper}\) by (metis of-nat-less-iff of-nat-power [symmetric])
{
fix \(m\)
assume \(m: \text{prime}\ m\) multiplicity \(m\ n > 0\)
hence \(m: m \in \text{prime-factors}\ n\)
by (auto simp: prime-factors-multiplicity)
hence \(m: m \mid n\)
by auto
then obtain \(k\) where \(n: n = m \ast k\ ..
from \(mp\) have \(pm: \text{prime}\ m\) by auto
hence \(m: m \geq 2\) and \(m: m > 0\) by (auto simp: prime-nat-iff)
from prems(6)[OF m2] md have mi: \(m \geq i\) by force
{
assume multiplicity \(m\ n = 1\)
with \(m\) have \(\exists k.\ \text{multiplicity}\ m\ n = 2 \ast k\) by presburger
then obtain \(j\) where \(\text{mult: multiplicity}\ m\ n = 2 \ast j\ ..
from \(n0\) have \(k: k > 0\) by auto
from \(\text{mult}\ m0\ k\ n\ m\) have \(\text{multiplicity}\ m\ k > 0\)
by (auto simp: prime-elem-multiplicity-mult-distrib)
with \(m\) have \(mp: m \in \text{prime-factors}\ k\)
by (auto simp: prime-factors-multiplicity)
hence \(md: m \mid d\ k\) by (auto simp: k)
then obtain \(l\) where \(kml: k = m \ast l\ ..
note \(n = n[\text{unfolded}\ kml]\)
from \(n\) have \(l \mid d\ n\) by auto
with prems(6)[of \(l\)] have \(l \leq 1 \lor l \geq i\) by arith
with \( n \) \( \mathsf{n0} \) have \( l : l = 1 \lor l \geq i \) by auto
from \( n \) \( \text{prems}(\ell) \) have \( l \neq 1 \) by auto
with \( l \) have \( l \geq i \) by auto
from \( \text{mult-le-mono}[\text{OF mult-le-mono}[\text{OF } mi \ mi] \ \ell] \) have \( n \geq i^3 \) unfolding \( n \) by (auto simp: power3_eq_cube)
with \( n-i\mathsf{3} \) have \( \text{False} \) by auto
}

with \( mi \) \( m \) have \( \text{multiplicity } m \ n = 1 \land m \geq i \) by auto
\[
\text{also have } \ldots \leq 1
\]
proof (cases \( m < i \))
  case True
  from \( \text{prems}(7)[\text{OF } m] \ n[\text{of } m] \ True \ m \) show \( \text{thesis} \) by force
next
  case False
  hence \( m \geq i \) by auto
  from \( \text{prems}(8)[\text{OF } m(1) \ \text{this}] \ n[\text{of } m] \ m \) show \( \text{thesis} \) by force
qed

finally show \( \text{multiplicity } n \ p \leq 1 \).
qed

definition \( \text{prime-product-factor} :: \mathsf{nat} \Rightarrow \mathsf{nat} \times \mathsf{nat} \) where
\[
\text{prime-product-factor } n = (\text{case } \sqrt{-\mathsf{nat}} \ n \ of
  \begin{array}{l}
  (\text{Cons } s -) \Rightarrow (s,1) \\
  [] \Rightarrow \text{prime-product-factor-main } 1 \ 1 (\mathsf{nat} \ \text{(root-nat-floor } 3 \ n)) \ n \ 2)
\end{array}
\]

lemma \( \text{prime-product-one}[\text{simp, intro}]: \text{prime-product } 1 \)
unfolding \( \text{prime-product-def } \) multiplicity-one-nat by auto

lemma \( \text{prime-product-factor: assumes } pf: \text{prime-product-factor } n = (sq,p) \)
shows \( n = sq \ast sq \ast p \land \text{prime-product } p \)
proof (cases \( \sqrt{-\mathsf{nat}} \ n \))
  case (Cons \( s \))
  from \( pf[\text{unfolded prime-product-factor-def Cons}] \ \text{arg-cong}[\text{OF } \text{Cons}, \ \text{of set}] \)
  \( \text{prime-product-one} \)
  show \( \text{thesis} \) by auto
next
  case \( \text{Nil} \)
  from \( \text{arg-cong}[\text{OF } \text{Nil}, \ \text{of set}] \) have \( \text{nsq} : \neg (\exists s. s \ast s = n) \) by auto
  show \( \text{thesis} \)
  by (rule \( \text{prime-product-factor-main}[\text{OF } \text{nsq refl, of } 1 \ 1 \ 2] \), unfold multiplicity-one-nat)

\]
4 A representation of real numbers via triples

theory Real_Impl
imports Sqrt-Babylonian begin

We represent real numbers of the form \( p + q \cdot \sqrt{b} \) for \( p, q \in \mathbb{Q}, n \in \mathbb{N} \) by triples \((p, q, b)\). However, we require the invariant that \( \sqrt{b} \) is irrational. Most binary operations are implemented via partial functions where the common the restriction is that the numbers \( b \) in both triples have to be identical. So, we support addition of \( \sqrt{2} + \sqrt{2} \), but not \( \sqrt{2} + \sqrt{3} \).

The set of natural numbers whose sqrt is irrational

\[ \text{definition } \text{sqrt-irrat} = \{ q :: \text{nat.} \; \neg \exists p. p \cdot p = \text{rat-of-nat } q \} \]

\[ \text{lemma } \text{sqrt-irrat}: \text{assumes choice: } q = 0 \lor b \in \text{sqrt-irrat} \]
\[ \text{and eq: real-of-rat } p + \text{real-of-rat } q \cdot \sqrt{\text{of-nat } b} = 0 \]
\[ \text{shows } q = 0 \]
\[ \text{using choice} \]
\[ \text{proof (cases } q = 0) \]
\[ \text{case False with choice have } \sqrt{\text{of-nat } b} : b \in \text{sqrt-irrat} \text{ by blast} \]
\[ \text{from eq have } \text{real-of-rat } q \cdot \sqrt{\text{of-nat } b} = \text{real-of-rat } (-p) \]
\[ \text{by (auto simp: of-rat-minus)} \]
\[ \text{then obtain } p \text{ where real-of-rat } q \cdot \sqrt{\text{of-nat } b} = \text{real-of-rat } p \text{ by blast} \]
\[ \text{from ary-cong[OF this, of } \lambda x. x \cdot x \text{ have real-of-rat } (q \cdot q) \cdot (\sqrt{\text{of-nat } b}) \]
\[ \text{real-of-rat } (p \cdot p) \text{ by (auto simp: field-simps of-rat-mult)} \]
\[ \text{also have } \sqrt{\text{of-nat } b} \cdot \sqrt{\text{of-nat } b} = \text{of-nat } b \text{ by simp} \]
\[ \text{finally have } \text{real-of-rat } (q \cdot q \cdot \text{rat-of-nat } b) = \text{real-of-rat } (p \cdot p) \text{ by (auto simp: of-rat-mult)} \]
\[ \text{hence } q \cdot q \cdot (\text{rat-of-nat } b) = p \cdot p \text{ by auto} \]
\[ \text{from ary-cong[OF this, of } \lambda x. x \cdot (q \cdot q) \text{]} \]
\[ \text{have } (p / q) \cdot (p / q) = \text{rat-of-nat } b \text{ using False by (auto simp: field-simps)} \]
\[ \text{with sqrt-irrat show } \text{thesis unfolding sqrt-irrat-def by blast} \]

qed

To represent numbers of the form \( p + q \cdot \sqrt{b} \), use mini algebraic numbers, i.e., triples \((p, q, b)\) with irrational \( \sqrt{b} \).

typedef mini-alg =
\[ \{ (p,q,b) \mid (p :: \text{rat}) \cdot (q :: \text{rat}) \cdot (b :: \text{nat}) \} \]
\[ q = 0 \lor b \in \text{sqrt-irrat} \]
by auto

setup-lifting type-definition-mini-alg

lift-definition real-of :: mini-alg ⇒ real is
  λ (p,q,b). of-rat p + of-rat q * sqrt (of-rat b) .

lift-definition ma-of-rat :: rat ⇒ mini-alg is λ x. (x,0,0) by auto

lift-definition ma-rat :: mini-alg ⇒ rat is fst .

lift-definition ma-base :: mini-alg ⇒ nat is snd o snd .

lift-definition ma-coeff :: mini-alg ⇒ rat is fst o snd .

lift-definition ma-auminus :: mini-alg ⇒ mini-alg is
  λ (p1,q1,b1). (− p1, − q1, b1) by auto

lift-definition ma-compatible :: mini-alg ⇒ mini-alg ⇒ bool is
  λ (p1,q1,b1) (p2,q2,b2). q1 = 0 ∨ q2 = 0 ∨ b1 = b2 .

definition ma-normalize :: rat × rat × nat ⇒ rat × rat × nat where
  ma-normalize x ≡ case x of (a,b,c) ⇒ if b = 0 then (a,0,0) else (a,b,c)

lemma ma-normalize-case[simp]: (case ma-normalize r of (a,b,c) ⇒ real-of-rat a + real-of-rat b * sqrt (of-rat c))
  = (case r of (a,b,c) ⇒ real-of-rat a + real-of-rat b * sqrt (of-rat c))
by (cases r, auto simp: ma-normalize-def)

lift-definition ma-plus :: mini-alg ⇒ mini-alg ⇒ mini-alg is
  λ (p1,q1,b1) (p2,q2,b2). if q1 = 0 then
    (p1 + p2, q2, b2) else ma-normalize (p1 + p2, q1 + q2, b1) by (auto simp: ma-normalize-def)

lift-definition ma-times :: mini-alg ⇒ mini-alg ⇒ mini-alg is
  λ (p1,q1,b1) (p2,q2,b2). if q1 = 0 then
    ma-normalize (p1*p2, p1*q2, b2) else
    ma-normalize (p1*p2 + of-nat b2*q1*q2, p1*q2 + q1*p2, b1) by (auto simp: ma-normalize-def)

lift-definition ma-inverse :: mini-alg ⇒ mini-alg is
  λ (p,q,b). let d = inverse (p * p − of-nat b * q * q) in
  ma-normalize (p * d, − q * d, b) by (auto simp: Let-def ma-normalize-def)

lift-definition ma-floor :: mini-alg ⇒ int is
  λ (p,q,b). case (quotient-of p,quotient-of q) of ((z1,n1),(z2,n2)) ⇒
    let z2n1 = z2 * n1; z1n2 = z1 * n2; n12 = n1 * n2; prod = z2n1 * z2n1 * int b in
    (z1n2 + (if z2n1 ≥ 0 then sqrt-int-floor-pos prod else − sqrt-int-ceiling-pos prod)) div n12 .
**lift-definition** \( \text{ma-sqrt} :: \text{mini-alg} \Rightarrow \text{mini-alg} \text{ is} \)
\[
\begin{align*}
\lambda (p,q,b). \text{let } (a,b) &= \text{quotient-of } p;\ aa = \text{abs } (a * b) \text{ in} \\
\text{case } \text{sqrt-int } aa \text{ of } [] &\Rightarrow (0, \text{inverse } (\text{of-int } b), \text{nat } aa) \mid (\text{Cons } s -) \Rightarrow (\text{of-int } s / \text{of-int } b, 0, 0)
\end{align*}
\]
**proof (unfold Let-def)**
\[
\begin{align*}
\text{fix } \text{prod} :: \text{rat } \times \text{rat } \times \text{nat} \\
\text{obtain } p \ q \ b \ \text{where } \text{prod: } \text{prod} = (p,q,b) \ \text{by } (\text{cases } \text{prod}, \text{auto}) \\
\text{obtain } a \ b \ \text{where } p: \text{quotient-of } p = (a,b) \ \text{by } \text{force} \\
\text{show } \exists p \ q \ b. (\text{case } \text{prod of} \\
(p, q, b) \Rightarrow \\
\text{case } \text{quotient-of } p \ of \\
(a, b) \Rightarrow \\
(\text{case } \text{sqrt-int } |a * b| \text{ of } [] \Rightarrow (0, \text{inverse } (\text{of-int } b), \text{nat } |a * b|) \\
| s \ # x \Rightarrow (\text{of-int } s / \text{of-int } b, 0, 0)) = \\
(p, q, b) \land \\
(q = 0 \lor b \in \text{sqrt-irrat})
\end{align*}
\]
**proof (cases \text{sqrt-int } (\text{abs } (a * b)))**
\[
\begin{align*}
\text{case } \text{Nil} \\
\text{from } \text{sqrt-int } (\text{abs } (a * b)) \ \text{Nil have } \text{irrat: } \neg (\exists \ y. \ y * y = |a * b|) \ \text{by } \text{auto} \\
\text{have } \text{nat } |a * b| \in \text{sqrt-irrat} \\
\text{proof (rule ccontr)} \\
\text{assume } \text{nat } |a * b| \notin \text{sqrt-irrat} \\
\text{then obtain } x :: \text{rat} \\
\text{where } x * x = \text{rat-of-nat } (\text{nat } |a * b|) \ \text{unfolding } \text{sqrt-irrat-def} \ \text{by } \text{auto} \\
\text{hence } x * x = \text{rat-of-int } |a * b| \ \text{by } \text{auto} \\
\text{from } \text{sqrt-rat-of-int}[\text{OF this}] \ \text{irrat} \ \text{show } \text{False} \ \text{by } \text{blast} \\
\text{qed} \\
\text{thus } \text{thesis using } \text{Nil by } (\text{auto simp: } \text{prod } p) \\
\text{qed (auto simp: } \text{prod } p \ \text{Cons})
\end{align*}
\]
**lift-definition** \( \text{ma-equal} :: \text{mini-alg} \Rightarrow \text{mini-alg} \Rightarrow \text{bool is} \)
\[
\begin{align*}
\lambda (p1,q1,b1) (p2,q2,b2). \\
p1 = p2 \land q1 = q2 \land (q1 = 0 \lor b1 = b2).
\end{align*}
\]
**lift-definition** \( \text{ma-ge-0} :: \text{mini-alg} \Rightarrow \text{bool is} \)
\[
\begin{align*}
\lambda (p,q,b). \text{let } bqq = \text{of-nat } b * q * q; pp = p * p \text{ in} \\
0 \leq p \land bqq \leq pp \lor 0 \leq q \land pp \leq bqq.
\end{align*}
\]
**lift-definition** \( \text{ma-is-rat} :: \text{mini-alg} \Rightarrow \text{bool is} \)
\[
\begin{align*}
\lambda (p,q,b). \ q = 0.
\end{align*}
\]
**definition** \( \text{ge-0 :: real } \Rightarrow \text{bool where } [\text{code def}]: \text{ge-0 } x = (x \geq 0) \)
\[
\begin{align*}
\text{lemma } \text{ma-ge-0}: \text{ge-0 } (\text{real-of } x) = \text{ma-ge-0 } x \\
\text{proof (transfer, unfold Let-def, clarsimp)} \\
\text{fix } p' q' :: \text{rat and } b' :: \text{nat} \\
\text{assume } b': q' = 0 \lor b' \in \text{sqrt-irrat} \\
\text{define } b \ \text{where } b = \text{real-of-nat } b'
\end{align*}
\]

proof
lemma ma-inverse
from b' have b: 0 ≤ b q = 0 ∨ b' ∈ sqrt-irrat unfolding b-def q-def by auto
define qb where qb = q * sqrt b
from b have sqrt: sqrt b ≥ 0 by auto
from b(2) have disj: q = 0 ∨ b ≠ 0 unfolding sqrt-irrat-def b-def by auto
have bdef: b = real-of-rat (of-nat b') unfolding b-def by auto
have (0 ≤ p + q * sqrt b) = (0 ≤ p + qb) unfolding qb-def by simp
also have ... ⇐⇒ (0 ≤ p ∧ abs qb ≤ abs p ∨ 0 ≤ q ∧ abs qb ≤ abs qb) by arith
also have ... ⇐⇒ (0 ≤ p ∧ qb ≤ p ∧ p ∨ 0 ≤ qb ∧ p ∨ p ≤ qb * qb)
  unfolding abs-leseq-square..
also have qb * qb = b * q * q unfolding qb-def
using b by auto
also have 0 ≤ qb ⇐⇒ 0 ≤ q unfolding qb-def using sqrt disj
  by (metis le-cases mult-eq-0-iff mult-antisym mult-nonpos-nonneg order-class.order.antisym qb-def real-sqrt-eq-zero-cancel-iff)
also have (0 ≤ p ∧ b * q * q ≤ p * p ∨ 0 ≤ q ∧ p * p ≤ b * q * q)
  ⇐⇒ (0 ≤ p' ∧ of-nat b' * q' * q' ≤ p' * p' ∨ 0 ≤ q' ∧ p' * p' ≤ of-nat b' * q'* q')
  unfolding qb-def
  by (unfold bdef p-def q-def of-rat-mul[order-class] of-rat-leseq simp)
finally
show ge-0 (real-of-rat p' + real-of-rat q' * sqrt (of-nat b')) =
(0 ≤ p' ∧ of-nat b' * q' * q' ≤ p' * p' ∨ 0 ≤ q' ∧ p' * p' ≤ of-nat b' * q'* q')
  unfolding ge-0-def p-def b-def q-def
  by (auto simp: of-rat-add of-rat-mult)
qed

lemma ma-0: 0 = real-of (ma-of-rat 0) by (transfer, auto)

lemma ma-1: 1 = real-of (ma-of-rat 1) by (transfer, auto)

lemma ma-uminus:
  (real-of x) = real-of (ma-uminus x)
  by (transfer, auto simp: of-rat-minus)

lemma ma-inverse: inverse (real-of r) = real-of (ma-inverse r)
proof (transfer, unfold Let-def, clarsimp)
fix p' q' :: rat and b' :: nat
assume b': q' = 0 ∨ b' ∈ sqrt-irrat
define b where b = real-of-rat b'
define p where p = real-of-rat p'
define q where q = real-of-rat q'
from b' have b: b ≥ 0 q = 0 ∨ b' ∈ sqrt-irrat unfolding b-def q-def by auto
have inverse (p + q * sqrt b) = (p - q * sqrt b) * inverse (p - p * b * (q + q))
proof (cases q = 0)
case True thus ?thesis by (cases p = 0, auto simp: field-simps)
next
case False
from sqrt-irrat[OF b', of p'] b-def p-def q-def False have mnull: p + q * sqrt b

12
≠ 0 by auto

have ?thesis ⟷ (p + q * sqrt b) * inverse (p + q * sqrt b) =
    (p + q * sqrt b) * ((p - q * sqrt b) * inverse (p * p - b * (q * q)))
unfolding mult-left-cancel[OF nnull] by auto
also have (p + q * sqrt b) * inverse (p + q * sqrt b) = 1 using nnull by auto
also have (p + q * sqrt b) * (p - q * sqrt b) * inverse (p * p - b * (q * q))
         = (p * p - b * (q * q)) * inverse (p * p - b * (q * q))
using b by (auto simp: field-simps)
also have ... = 1
proof (rule right-inverse, rule)
  assume eq: p * p - b * (q * q) = 0
  have real-of-rat (p' * p' / (q' * q')) = p * p / (q * q)
    unfolding p-def b-def q-def by (auto simp: of-rat-mult of-rat-divide)
  also have ... = b using eq False by (auto simp: field-simps)
also have ... = real-of-rat (of-nat b') unfolding b-def by auto
finally have (p' / q') * (p' / q') = of-nat b'
unfolding of-rat-eq-iff by simp
with b False show False unfolding sqrt-irrat-def by blast
qed

finally
show ?thesis by simp

qed

thus inverse (real-of-rat p' + real-of-rat q' * sqrt (of-nat b')) =
    real-of-rat (p' * inverse (p' * p' - of-nat b' * q' * q')) +
    real-of-rat (− (q' * inverse (p' * p' - of-nat b' * q' * q'))) * sqrt (of-nat b')
by (simp add: divide-simps of-rat-mult of-rat-divide of-rat-diff of-rat-minus b-def
p-def q-def
    split: if-splits)

qed

lemma ma-sqrt-main: ma-rat r ≥ 0 ⟹ ma-coeff r = 0 ⟹ sqrt (real-of r) =
    real-of (ma-sqrt r)
proof (transfer, unfold Let-def, clarsimp)
  fix p :: rat
  assume p: 0 ≤ p
  hence abs: abs p = p by auto
  obtain a b where ab: quotient-of p = (a,b) by force
  hence pab: p = of-int a / of-int b by (rule quotient-of-denom-pos)
  from ab have b: b > 0 by (rule quotient-of-denom-pos)
  with p pab have abpos: a * b ≥ 0
    by (metis of-int-0-le-iff of-int-le-0-iff zero-le-divide-iff zero-le-mult-iff)
  have rabs: of-nat (nat (a * b)) = real-of-int a * real-of-int b using abpos
    by simp
  let ?lhs = sqrt (of-int a / of-int b)
  let ?rhs = (case case quotient-of p of
             (a, b) ⇒ (case sqrt-int [a * b] of [] ⇒ (0, inverse (of-int b), nat |a * b|)
               | s # x ⇒ (of-int s / of-int b, 0, 0)) of
             (p, q, b) ⇒ of-rat p + of-rat q * sqrt (of-nat b))
  have sqrt (real-of-rat p) = ?lhs unfolding pab
by (metis of-rat-divide of-rat-of-int-eq)
also have \ldots = \?rhs
proof (cases sqrt-int \[a * b\])
case Nil
define sb where \(sb = \sqrt{(\text{of-int} \ b)}\)
define sa where \(sa = \sqrt{(\text{of-int} \ a)}\)
from \(b \ \text{sb-def}\) have \(sb > 0\) \(\text{real-of-int} \ b > 0\) by auto
have sbb: \(sb * sb = \text{real-of-int} \ b\) unfolding sb-def
  by (rule sqrt-sqrt, insert b, auto)
from Nil have \(?\text{thesis} = (sa / sb = \frac{1}{\text{of-int} \ b} * (sa * sb))\) unfolding ab using abpos
  by (metis b divide-real-def eq-divide-imp inverse-divide inverse-inverse-eq
    inverse-mult-distrib less-int-code (1) of-int-eq-0-iff real-sqrt-eq-zero-cancel-iff sb-def
    sbb times-divide-eq-right)
also have \(\ldots = \text{True}\) using sb
finally show \(?\text{thesis} by simp\)
next
case (Cons s x)
from \(b\) have \(b > 0\) by auto
from Cons sqrt-int \[\text{of abs} (a * b)\] have \(s * s = \text{abs} (a * b)\) by auto
with \(\text{sqrt-int-pos} (\text{OF} \ \text{Cons})\) have \(\text{sqrt} (\text{real-of-int} (\text{abs} (a * b))) = \text{of-int} \ s\)
  by (metis abs-of-nonneg of-int-mult of-int-abs real-sqrt-abs2
    symmetric)
with \text{abpos} have \(\text{of-int} \ s = \sqrt{(\text{real-of-int} (a * b))}\) by auto
thus \(?\text{thesis}\) unfolding ab split using Cons b
  by (auto simp: of-rat-divide field-simps real-sqrt-divide real-sqrt-mult)
qed
finally show \(\text{sqrt} (\text{real-of-rat} \ p) = \?\text{rhs}\).
qed

lemma ma-sqrt: \(\text{sqrt} (\text{real-of} \ r) = (\text{if} \ \text{ma-coeff} \ r = 0 \text{ then } \text{if} \ \text{ma-rat} \ r \geq 0 \text{ then real-of } (\text{ma-sqrt} \ (\text{ma-uminus} \ r)) \text{ else } \text{real-of } (\text{ma-sqrt} \ (\text{ma-uminus} \ r)))\)
  else Code.abort (STR "cannot represent sqrt of irrational number") (\lambda -. \text{sqrt (real-of r)})
proof (cases \text{ma-coeff} \ r = 0)
case True note 0 = this
  hence 00: \(\text{ma-coeff} (\text{ma-uminus} \ r) = 0\) by (transfer, auto)
  show \(?\text{thesis}\)
  proof (cases \text{ma-rat} \ r \geq 0)
    case True
    from ma-sqrt-main[OF this 0] 0 True show \(?\text{thesis}\) by auto
  next
    case False
    hence \(\text{ma-rat} (\text{ma-uminus} \ r) \geq 0\) by (transfer, auto)
    from ma-sqrt-main[OF this 00, folded ma-uminus, symmetric] False 0
    show \(?\text{thesis}\) by (auto simp: real-sqrt-minus)
  qed
qed
qed auto
lemma ma-plus:
  (real-of r1 + real-of r2) = (if ma-compatible r1 r2
  then real-of (ma-plus r1 r2) else
  Code.abort (STR "different base") (λ -. real-of r1 + real-of r2))
by transfer (auto split: prod.split simp: field-simps of-rat-add)

lemma ma-times:
(\text{real-of } r1 \ast \text{real-of } r2) = (if ma-compatible r1 r2
then \text{real-of } (\text{ma-times r1 r2}) else
Code.abort (STR "different base") (λ -. \text{real-of } r1 \ast \text{real-of } r2))
by transfer (auto split: prod.split simp: field-simps of-rat-mult of-rat-add)

lemma ma-equal:
  HOL.equal (real-of r1) (real-of r2) = (if ma-compatible r1 r2
then ma-equal r1 r2 else
Code.abort (STR "different base") (λ -. HOL.equal (real-of r1) (real-of r2)))
proof (transfer, unfold equal-real-def, clarsimp)

fix p1 q1 p2 q2 :: rat and b1 b2 :: nat
assume b1: q1 = 0 ∨ b1 ∈ sqrt-irrat
assume b2: q2 = 0 ∨ b2 ∈ sqrt-irrat
assume base: q1 = 0 ∨ q2 = 0 ∨ b1 = b2
let ?l = real-of-rat p1 + real-of-rat q1 * sqrt (of-nat b1) =
real-of-rat p2 + real-of-rat q2 * sqrt (of-nat b2)
let ?m = real-of-rat q1 * sqrt (of-nat b1) = real-of-rat (p2 - p1) + real-of-rat q2
* sqrt (of-nat b2)
let ?r = p1 = p2 ∧ q1 = q2 ∧ (q1 = 0 ∨ b1 = b2)

have ?l ←→ real-of-rat q1 * sqrt (of-nat b1) =
real-of-rat (p2 - p1) + real-of-rat q2
* sqrt (of-nat b2)
  by (auto simp: of-rat-add of-rat-diff of-rat-minus)
also have . . . ←→ p1 = p2 ∧ q1 = q2 ∧ (q1 = 0 ∨ b1 = b2)
proof
  assume ?m
  from base have q1 = 0 ∨ q1 ≠ 0 ∧ q2 = 0 ∨ q1 ≠ 0 ∧ q2 ≠ 0 ∧ b1 = b2 by auto
  thus p1 = p2 ∧ q1 = q2 ∧ (q1 = 0 ∨ b1 = b2)
  proof
    assume q1: q1 = 0
    with ⟨?m⟩ have real-of-rat (p2 - p1) + real-of-rat q2 * sqrt (of-nat b2) = 0
      by auto
  with sqrt-irrat b2 have q2: q2 = 0 by auto
  with q1 ⟨?m⟩ show ?thesis by auto
next
  assume q1 ≠ 0 ∧ q2 = 0 ∨ q1 ≠ 0 ∧ q2 ≠ 0 ∧ b1 = b2
  thus ?thesis
  proof
    assume ass: q1 ≠ 0 ∧ q2 = 0
    with ⟨?m⟩ have real-of-rat (p1 - p2) + real-of-rat q1 * sqrt (of-nat b1) = 0
      by (auto simp: of-rat-diff)
with \( b1 \) have \( q1 = 0 \) using sqrt-irrat by auto

with \( \text{ass} \) show \( \text{thesis} \) by auto

next

assume \( \text{ass}: q1 \neq 0 \land q2 \neq 0 \land b1 = b2 \)

with \( \langle ?m \rangle \) have \( \ast: \text{real-of-rat} \ (p2 - p1) + \text{real-of-rat} \ (q2 - q1) \ast \text{sqrt} \ (\text{of-nat} \ b2) = 0 \)

by (auto simp: field-simps of-rat-diff)

have \( q2 - q1 = 0 \)

by (rule sqrt-irrat[OF \(- \ast\), insert \( \text{ass} \) \( \text{b2} \), auto])

with \( \ast \) \( \text{ass} \) show \( \text{thesis} \) by auto

qed

qed

auto

finally show \( \text{id} = \text{id} \) by simp

qed

lemma \( \text{ma-floor}: \text{floor} \ (\text{real-of} \ r) = \text{ma-floor} \ r \)

proof (transfer, unfold \text{Let-def}, clarsimp)

fix \( p :: \text{rat} \) and \( b :: \text{nat} \)

obtain \( z1 \ n1 \) where \( \text{qp: quotient-of} \ p = (z1, n1) \) by force

obtain \( z2 \ n2 \) where \( \text{qq: quotient-of} \ q = (z2, n2) \) by force

from quotient-of-denom-pos[OF \( \text{qp} \)] have \( n1: 0 < n1 \).

from quotient-of-denom-pos[OF \( \text{qq} \)] have \( n2: 0 < n2 \).

from quotient-of-div[OF \( \text{qp} \)] have \( p: p = \text{of-int} \ z1 / \text{of-int} \ n1 \).

from quotient-of-div[OF \( \text{qq} \)] have \( q: q = \text{of-int} \ z2 / \text{of-int} \ n2 \).

have \( p: p = \text{of-int} \ (z1 \ast n2) / \text{of-int} \ (n1 \ast n2) \) unfolding \( p \) using \( n2 \) by auto

have \( q: q = \text{of-int} \ (z2 \ast n1) / \text{of-int} \ (n1 \ast n2) \) unfolding \( q \) using \( n1 \) by auto

define \( z1n2 \) where \( z1n2 = z1 \ast n2 \)

define \( z2n1 \) where \( z2n1 = z2 \ast n1 \)

define \( n12 \) where \( n12 = n1 \ast n2 \)

define \( r\text{-add} \) where \( r\text{-add} = \text{of-int} \ (z1n2) \ast \text{sqrt} \ (\text{real-of-int} \ (\text{int} \ b)) \)

from \( \text{n1 n2 have} \ n120: n12 > 0 \) unfolding \( \text{n12-def} \) by simp

have \( \text{floor} \ (\text{of-rat} \ p + \text{of-rat} \ q \ast \text{sqrt} \ (\text{real-of-nat} \ b)) = \text{floor} \ ((\text{of-int} \ z1n2 + \text{r-add}) / \text{of-int} \ n12) \)

unfolding \( r\text{-add-def} \text{n12-def} z1n2-def \text{z2n1-def} \)

unfolding \( p \) \( q \) \text{add-divide-distrib} \text{of-rat-divide} \text{of-rat-of-int-eq} \text{of-int-of-nat-eq} \text{by simp} \)

also have \( \ldots = \text{floor} \ (\text{of-int} \ z1n2 + \text{r-add}) \text{ div} \ n12 \)

by (rule \text{floor-div-pos-int}[OF \( \text{n120} \)])

also have \( \text{of-int} \ z1n2 + \text{r-add} = \text{of-int} \ z1n2 \text{ by simp} \)

also have \( \text{floor} \ (\ldots) = \text{floor} \text{ r-add} + z1n2 \text{ by simp} \)

also have \( \ldots = z1n2 + \text{floor} \text{ r-add by simp} \)

finally have \( \text{id: [of-rat} p + \text{of-rat} q \ast \text{sqrt} \ (\text{of-nat} \ b)] = (z1n2 + \lfloor \text{r-add} \rfloor) \text{ div} \ n12 \).

show \( \text{[of-rat} p + \text{of-rat} q \ast \text{sqrt} \ (\text{of-nat} \ b)] = \)

\( \langle \text{case quotient-of} \ p \text{ of} \)

\( (z1, n1) \Rightarrow \)

\( \text{case quotient-of} \ q \text{ of} \)

\( (z2, n2) \Rightarrow \)

16
(z1 * n2 + (if 0 ≤ z2 * n1 then sqrt-int-floor-pos (z2 * n1 * (z2 * n1) * int b) else sqrt-int-ceiling-pos (z2 * n1 * (z2 * n1) * int b))) div (n1 * n2))

unfolding q q split id n12-def z1n2-def
proof (rule arg-cong[of - - λ x. ((z1 * n2) + x) div (n1 * n2)])
have ge-int: z2 * n1 * (z2 * n1) * int b ≥ 0
  by (metis mult-nonneg-nonneg zero-le-square of-nat-0-le-iff)
show ⌊r-add⌋ = (if 0 ≤ z2 * n1 then sqrt-int-floor-pos (z2 * n1 * (z2 * n1) * int b) else sqrt-int-ceiling-pos (z2 * n1 * (z2 * n1) * int b))
proof (cases z2 * n1 ≥ 0)
case True
  hence ge: real-of-int (z2 * n1) ≥ 0 by (metis of-int-0-le-iff)
  have radd: r-add = sqrt (of-int (z2 * n1 * (z2 * n1) * int b))
    unfolding r-add-def z2n1-def using sqrt-sqrt[OF ge]
    by (simp add: ac-simps real-sqrt-mult)
  show ?thesis unfolding radd sqrt-int-floor-pos [OF ge-int] using True by simp
next
  case False
  hence ge: real-of-int (− (z2 * n1)) ≥ 0
    by (metis mult-zero-left neg-0-le-iff-le of-int-0-le-iff order-refl zero-le-mult-iff)
  have radd: r-add = − sqrt (of-int (z2 * n1 * (z2 * n1) * int b))
    unfolding r-add-def z2n1-def using sqrt-sqrt[OF ge]
    by (metis minus-minus minus-mult-commute minus-mult-right of-int-minus of-int-mult real-sqrt-minus real-sqrt-mult z2n1-def)
  hence radd: floor r-add = − ceiling (sqrt (of-int (z2 * n1 * (z2 * n1) * int b)))
    by (metis ceiling-def minus-minus)
  show ?thesis unfolding radd sqrt-int-ceiling-pos [OF ge-int] using False by simp
  qed
qed

lemma comparison-impl:
  (x :: real) ≤ (y :: real) = ge-0 (y − x)
  (x :: real) < (y :: real) = (x ≠ y ∧ ge-0 (y − x))
  by (simp-all add: ge-0-def, linarith+)

lemma ma-of-rat: real-of-rat r = real-of (ma-of-rat r)
  by (transfer, auto)

definition is-rat :: real ⇒ bool where
  [code-abbrev]: is-rat x ↔ x ∈ ℚ

lemma ma-is-rat: is-rat (real-of x) = ma-is-rat x
proof (transfer, unfold is-rat-def, clarsimp)
  fix p q :: rat and b :: nat
  let ?p = real-of-rat p

17
let \( q = \text{real-of-rat} \ q \)
let \( b = \text{real-of-nat} \ b \)
let \( b' = \text{real-of-rat} \ (\text{of-nat} \ b) \)
assume \( b : \ q = 0 \lor b \in \text{sqrt-irrat} \)
show \( (\ ?p + ?q \ast \sqrt ?b \in \mathbb{Q}) = (q = 0) \)
proof (cases \( q = 0 \))
case False
from False b have \( b : b \in \text{sqrt-irrat} \) by auto
{ assume \( \ ?p + \ ?q \ast \sqrt ?b \in \mathbb{Q} \)
from this \[\text{unfolded Rats-def} \] obtain \( r \) where \( r : \ ?p + \ ?q \ast \sqrt ?b = \text{real-of-rat} \ r \) by auto
let \( ?r = \text{real-of-rat} \ r \)
from \( r \) have \( \text{real-of-rat} (p - r) + \ ?q \ast \sqrt ?b = 0 \) by (simp add: of-rat-diff)
from sqrt-irrat [OF disjI2[OF b] this] False have False by auto
} thus \( ?\text{thesis} \) by auto
qed auto
qed

definition \( \text{sqrt-real} \ x = (\text{if} \ x \in \mathbb{Q} \ \land \ x \geq 0 \ \text{then} \ (\text{if} \ x = 0 \ \text{then} \ [0] \ \text{else} \ (\text{let} \ sx = \sqrt x \ \text{in} \ [sx, -sx])) \ \text{else} \ []) \)

lemma \( \text{sqrt-real[simp]} \)\[\text{assumes} \ x : x \in \mathbb{Q} \]
shows \( \text{set} (\text{sqrt-real} \ x) = \{ y . y * y = x \} \)
proof (cases \( x \geq 0 \))
case False
hence \( \land \ y . y * y \neq x \) by auto
with False show \( ?\text{thesis unfolding sqrt-real-def} \) by auto
next
case True
thus \( ?\text{thesis using} \ x \)
by (cases \( x = 0 \), auto simp: Let-def sqrt-real-def)
qed

code-datatype \( \text{real-of} \)
declare []
lemma []


end

5 A unique representation of real numbers via triples

typedef mini-alg-unique =
{ r :: mini-alg . ma-coeff r = 0 ∧ ma-base r = 0 ∨ ma-coeff r ≠ 0 ∧ prime-product (ma-base r) }
by (transfer, auto)

setup-lifting type-definition-mini-alg-unique

lift-definition real-of-u :: mini-alg-unique ⇒ real is real-of .
lift-definition mau-floor :: mini-alg-unique ⇒ int is ma-floor.
lift-definition mau-of-rat :: rat ⇒ mini-alg-unique is ma-of-rat by (transfer, auto)
lift-definition mau-rat :: mini-alg-unique ⇒ rat is ma-rat.
lift-definition mau-base :: mini-alg-unique ⇒ nat is ma-base.
lift-definition mau-coeff :: mini-alg-unique ⇒ rat is ma-coeff.
lift-definition mau-uminus :: mini-alg-unique ⇒ mini-alg-unique is ma-uminus by (transfer, auto)
lift-definition mau-compatible :: mini-alg-unique ⇒ mini-alg-unique ⇒ bool is ma-compatible.
lift-definition mau-ge-0 :: mini-alg-unique ⇒ bool is ma-ge-0.
lift-definition mau-inverse :: mini-alg-unique ⇒ mini-alg-unique is ma-inverse by (transfer, auto simp: ma-normalize-def Let-def split: if-splits)
lift-definition mau-plus :: mini-alg-unique ⇒ mini-alg-unique ⇒ mini-alg-unique is ma-plus by (transfer, auto simp: ma-normalize-def split: if-splits)
lift-definition mau-times :: mini-alg-unique ⇒ mini-alg-unique ⇒ mini-alg-unique is ma-times by (transfer, auto simp: ma-normalize-def split: if-splits)
lift-definition mau-equal :: mini-alg-unique ⇒ mini-alg-unique ⇒ bool is ma-identity.

lemma Ratreal-code[code]:
Ratreal = real-of-u ◦ mau-of-rat
by (simp add: fun-eq-iff) (transfer, transfer, simp)

lemma mau-floor: floor (real-of-u r) = mau-floor r
using ma-floor by (transfer, auto)

lemma mau-inverse: inverse (real-of-u r) = real-of-u (mau-inverse r)
using ma-inverse by (transfer, auto)

lemma mau-uminus: − (real-of-u r) = real-of-u (mau-uminus r)
using ma-uminus by (transfer, auto)

lemma mau-times:
(real-of-u r1 * real-of-u r2) = (if mau-compatible r1 r2
then real-of-u (mau-times r1 r2) else
Code.abort (STR "different base") (λ - real-of-u r1 * real-of-u r2))
using ma-times by (transfer, auto)

lemma mau-plus:
(real-of-u r1 + real-of-u r2) = (if mau-compatible r1 r2
then real-of-u (mau-plus r1 r2) else
Code.abort (STR "different base") (λ - real-of-u r1 + real-of-u r2))
using ma-plus by (transfer, auto)

lemma real-of-u-inj[simp]: real-of-u x = real-of-u y ⇔ x = y
proof
note field-simps[simp] of-rat-diff[simp]
assume real-of-u x = real-of-u y
thus x = y

20
proof (transfer)
\begin{align*}
\text{fix } x \text{ y} \\
\text{assume } \text{ma-coeff } x = 0 \land \text{ma-base } x = 0 \lor \text{ma-coeff } x \neq 0 \land \text{prime-product} \\
\text{(ma-base } x) \\
\text{and } \text{ma-coeff } y = 0 \land \text{ma-base } y = 0 \lor \text{ma-coeff } y \neq 0 \land \text{prime-product} \\
\text{(ma-base } y) \\
\text{and } \text{real-of } x = \text{real-of } y \\
\text{thus } x = y
\end{align*}

proof (transfer, clarsimp)
\begin{align*}
\text{fix } p1 \ q1 \ p2 \ q2 :: \text{rat and } b1 \ b2 \\
\text{let } ?p1 = \text{real-of-rat } p1 \\
\text{let } ?p2 = \text{real-of-rat } p2 \\
\text{let } ?q1 = \text{real-of-rat } q1 \\
\text{let } ?q2 = \text{real-of-rat } q2 \\
\text{let } ?b1 = \text{real-of-nat } b1 \\
\text{let } ?b2 = \text{real-of-nat } b2 \\
\text{assume } q1: q1 = 0 \land b1 = 0 \lor q1 \neq 0 \land \text{prime-product } b1 \\
\text{and } q2: q2 = 0 \land b2 = 0 \lor q2 \neq 0 \land \text{prime-product } b2 \\
\text{and } i1: q1 = 0 \lor b1 \in \sqrt{-\text{irrat}} \\
\text{and } i2: q2 = 0 \lor b2 \in \sqrt{-\text{irrat}} \\
\text{and eq: } ?p1 + ?q1 \cdot \sqrt{?b1} = ?p2 + ?q2 \cdot \sqrt{?b2} \\
\text{show } p1 = p2 \land q1 = q2 \land b1 = b2
\end{align*}

proof (cases q1 = 0)
\begin{align*}
\text{case True} \\
\text{have } q2 = 0 \\
\text{by } (\text{rule sqrt-irrat[OF i2, of } p2 - p1], \text{insert eq True } q1, \text{auto}) \\
\text{with True } q1 \ q2 \ eq \ \text{show ?thesis by auto}
\end{align*}

next
\begin{align*}
\text{case False} \\
\text{hence } 1: q1 \neq 0 \land \text{prime-product } b1 \ \text{using } q1 \ \text{by auto} \\
\{ \\
\text{assume } \ast: q2 = 0 \\
\text{have } q1 = 0 \\
\text{by } (\text{rule sqrt-irrat[OF i1, of } p1 - p2], \text{insert eq } q2, \text{auto}) \\
\text{with False have True by auto}
\}
\end{align*}

\begin{align*}
\text{hence } 2: q2 \neq 0 \land \text{prime-product } b2 \ \text{using } q2 \ \text{by auto} \\
\text{from } 1 \ i1 \ \text{have } b1: b1 \neq 0 \ \text{unfolding } \sqrt{-\text{irrat}-\text{def} by (cases } b1, \text{auto)} \\
\text{from } 2 \ i2 \ \text{have } b2: b2 \neq 0 \ \text{unfolding } \sqrt{-\text{irrat}-\text{def} by (cases } b2, \text{auto)} \\
\text{let } ?sq = \lambda x. x \cdot x \\
\text{define } q3 \ \text{where } q3 = p2 - p1 \\
\text{let } ?q3 = \text{real-of-rat } q3 \\
\text{let } ?e = \text{of-rat } (q2 * q2 * \text{of-nat } b2 + ?sq q3 - ?sq q1 * \text{of-nat } b1) + \text{of-rat} \\
(2 * q2 * q3) * \sqrt{?b2} \\
\text{from eq have } \ast: ?q1 * \sqrt{?b1} = ?q2 * \sqrt{?b2} + ?q3 \\
\text{by } (\text{simp add: q3-def}) \\
\text{from arg-cong[OF this, of } ?sq] \ \text{have } 0 = (\text{real-of-rat } 2 * ?q2 * ?q3) * \sqrt{?b2} + \\
(\text{?sq } ?q2 + ?b2 + \ ?sq ?q3 - \ ?sq ?q1 * \ ?b1)}
\end{align*}
ultimately obtain \( n_1 n_2 \) from \( b_2 = 0 \) \( b_1 \) from
moreover have
have
from
have
ultimately obtain
moreover have
ultimately obtain
ultimately obtain
moreover have
also have
let \( b_1 = \text{rat-of-nat} b_1 \)
let \( b_2 = \text{rat-of-nat} b_2 \)
from eq[unfolded \( q_3 \)] have eq: \( ?sq q_2 * ?b_2 = ?sq q_1 * ?b_1 \) by auto
obtain \( z_1 n_1 \) where \( d_1: \text{quotient-of} q_1 = (z_1, n_1) \) by force
obtain \( z_2 n_2 \) where \( d_2: \text{quotient-of} q_2 = (z_2, n_2) \) by force
note id = \( \text{quotient-of-div} \{OF \ d_1\} \ \text{quotient-of-div} \{OF \ d_2\} \)
note pos = \( \text{quotient-of-denom-pos} \{OF \ d_1\} \ \text{quotient-of-denom-pos} \{OF \ d_2\} \)
from id(1) I(1) pos(1) have \( z_1: z_1 \neq 0 \) by auto
from id(2) 2(1) pos(2) have \( z_2: z_2 \neq 0 \) by auto
let \( ?n_1 = \text{rat-of-int} n_1 \)
let \( ?n_2 = \text{rat-of-int} n_2 \)
let \( ?z_1 = \text{rat-of-int} z_1 \)
let \( ?z_2 = \text{rat-of-int} z_2 \)
from arg-cong[OF eq[simplified id], of \( \lambda \ x. x * \ ?sq \ ?n_1 * \ ?sq \ ?n_2, \ simplified \ field-simps\)]
have \( ?sq (\ ?n_1 * \ ?z_2) * \ ?b_2 = ?sq (\ ?n_2 * \ ?z_1) * \ ?b_1 \)
using pos by auto
moreover have \( \ ?n_1 * \ ?z_2 \neq 0 \ \ ?n_2 * \ ?z_1 \neq 0 \) using \( z_1 z_2 \) pos by auto
ultimately obtain \( i_1 i_2 \) where \( 0: \text{rat-of-int} i_1 \neq 0 \ \text{rat-of-int} i_2 \neq 0 \)
and eq: \( ?sq (\text{rat-of-int} i_2) * \ ?b_2 = ?sq (\text{rat-of-int} i_1) * \ ?b_1 \)
unfolding \( \text{of-int-mult\[symmetric\]} \) by blast+
let \( ?b_1 = \text{int} b_1 \)
let \( ?b_2 = \text{int} b_2 \)
from eq have eq: \( ?sq i_1 * \ ?b_1 = ?sq i_2 * \ ?b_2 \)
by (metis (hide-lams, no-types) \text{of-int-eq-iff of-int-mult of-int-of-nat-eq})
from \( 0 \) have \( 0: i_1 \neq 0 \ i_2 \neq 0 \) by auto
from arg-cong[OF eq, of nat] have \( ?sq (\text{nat (abs} i_1)) * b_1 = ?sq (\text{nat (abs} i_2)) * b_2 \)
by (metis \text{abs-of-nat eq nat-abs-mult-distrib nat-int})
moreover have \( \text{nat (abs} i_1) > 0 \ \text{nat (abs} i_2) > 0 \) using \( 0 \) by auto
ultimately obtain \( n_1 n_2 \) where \( 0: n_1 > 0 \ n_2 > 0 \) and eq: \( ?sq n_1 * b_1 = ?sq n_2 * b_2 \) by blast
from \( b_1 \) 0 have \( b_1: b_1 > 0 \ n_1 > 0 \ n_1 * n_1 > 0 \) by auto
from \( b_2 \) 0 have \( b_2: b_2 > 0 \ n_2 > 0 \ n_2 * n_2 > 0 \) by auto
{
fix \( p :: \text{nat assume} \ p: \text{prime} p \)
have multiplicity p (\( ?sq n_1 * b_1 \)) = multiplicity p b_1 + 2 * multiplicity p n_1
using b_1 p by (auto simp: prime-elem-multiplicity-mult-distrib)
also have \( \ldots \mod 2 = \text{multiplicity} p b_1 \mod 2 \) by presburger
finally have \( \text{id1: multiplicity} p (\( ?sq n_1 * b_1 \)) \mod 2 = \text{multiplicity} p b_1 \mod 2 \)

have multiplicity p (\( ?sq n_2 * b_2 \)) = multiplicity p b_2 + 2 * multiplicity p
using $b_2 \cdot p$ by (auto simp: prime-elem-multiplicity-mult-distrib)
also have \(...\) mod 2 = multiplicity $p \cdot b_2$ mod 2 by presburger
finally have $id_2$: multiplicity $p$ (?sq $n_2 \cdot b_2$) mod 2 = multiplicity $p \cdot b_2$
mod 2.
from $id_1 \cdot id_2 \cdot eq$ have $eq$: multiplicity $p \cdot b_1$ mod 2 = multiplicity $p \cdot b_2$ mod 2
by simp
from $1(2) \cdot 2(2) \cdot p$ have multiplicity $p \cdot b_1$ \leq 1 multiplicity $p \cdot b_2$ \leq 1
unfolding prime-product-def by auto
with $eq$ have multiplicity $p \cdot b_1 = \text{multiplicity } p \cdot b_2$ by simp
from $p \cdot q \cdot b$ show $\?thesis$ by blast
qed
qed
qed
simp

lift-definition mau-sqrt :: mini-alg-unique \Rightarrow mini-alg-unique is
\lambda ma. let $(a,b)$ = quotient-of (ma-rat ma); (sq,fact) = prime-product-factor (nat (abs $a \cdot b$));
$ma' = ma\cdot of\cdot rat (of\cdot int (sgn(a)) \cdot of\cdot nat sq \div of\cdot int b)$
in ma-times $ma'$ (ma-sqrt (ma-of-rat (of-nat fact)))

proof
fix $ma$ :: mini-alg
let $?num =$
let $(a, b) = \text{quotient-of (ma-rat ma)}$; (sq,fact) = prime-product-factor (nat (abs $a \cdot b$));
$ma' = ma\cdot of\cdot rat (\text{rat-of-int (sgn a)} \cdot \text{of-nat sq} \div \text{of-int b})$
in ma-times $ma'$ (ma-sqrt (ma-of-rat (of-nat fact)))

obtain $a \cdot b$ where $q$ = quotient-of (ma-rat ma) = $(a, b)$ by force
obtain sq fact where ppf: prime-product-factor (nat (abs $a \cdot b$)) = (sq,fact) by force

define asq where $asq = \text{rat-of-int (sgn a)} \cdot \text{of-nat sq} \div \text{of-int b}$
define $ma'$ where $ma' = ma\cdot of\cdot rat asq$
define sqrt where $sqrt = ma\cdot sqrt (ma\cdot of\cdot rat (\text{of-nat fact}))$

have $num$: $?num = ma\cdot times ma' \cdot sqrt$ unfolding $q$ ppf asq-def Let-def split $ma'\cdot def sqrt\-def ..$
let $?inv = \lambda ma. ma\cdot coeff ma = 0 \land ma\cdot base ma = 0 \lor ma\cdot coeff ma \neq 0 \land$
prime-product (ma-base ma)

have $ma'$: $?inv ma' \cdot unfolding$ $ma'\-def$
by (transfer, auto)

have id: \( i \cdot int i \cdot 1 = i \land i :: rat. i / 1 = i \cdot rat\-of\-int \cdot 1 = 1 \cdot inverse (1 :: rat) \)
= 1
\( \land n. \cdot nat |int n| = n \) by auto
from prime-product-factor[OF ppf] have prime-product fact by auto
hence sqrt: $?inv sqrt$ unfolding $sqrt\-def$
by (transfer, unfold split quotient-of-nat Let-def id, case-tac sqrt-int [int facta],
auto

show \texttt{?inv \texttt{?num unfolding num using ma’ sqrt}}
  \texttt{by (transfer, auto simp: ma-normalize-def split: if-splits)}
qed

\textbf{lemma sqrt-sgn[simp]}: \texttt{sqrt (of-int (sgn a)) = of-int (sgn a)}
\texttt{by (cases a \geq 0, cases a = 0, auto simp: real-sqrt-minus)}

\textbf{lemma mau-sqrt-main}: \texttt{mau-coeff r = 0 \implies sqrt (real-of-u r) = real-of-u (mau-sqrt r)}
\textbf{proof (transfer)}
  \texttt{fix r}
  \texttt{assume ma-coeff r = 0}
  \texttt{hence rr: real-of r = of-rat (ma-rat r) by (transfer, auto)}
  \texttt{obtain a b where q: quotient-of (ma-rat r) = (a,b) by force}
  \texttt{from quotient-of-div[OF q] have r: ma-rat r = of-int a / of-int b by auto}
  \texttt{from quotient-of-denom-pos[OF q] have b: b > 0 by auto}
  \texttt{obtain sq fact where ppf: prime-product-factor (nat (|a| * b)) = (sq, fact) by force}
  \texttt{from prime-product-factor[OF ppf] have ab: nat (|a| * b) = sq * sq * fact ..}
  \texttt{have \texttt{sqrt (real-of r) = sqrt(of-int a / of-int b) unfolding rr r}}
  \texttt{by (metis of-rat-divide of-rat-of-int-eq)}
  \texttt{also have real-of-int a / of-int b = of-int a * of-int b / (of-int b * of-int b) using b by auto}
  \texttt{also have sqrt (..) = sqrt (of-int a * of-int b) / of-int b using sqrt-sqrt[of real-of-int b] b}}
  \texttt{by (metis less-eq-real-def of-int-0-less-iff real-sqrt-divide real-sqrt-mult)}
\texttt{also have real-of-int a * of-int b = real-of-int (a * b) by auto}
  \texttt{also have a * b = sgn a * (abs a * b) by (simp, metis mult-sgn-abs)}
  \texttt{also have real-of-int (..) = of-int (sgn a) * real-of-int (|a| * b)}
  \texttt{unfolding of-int-mult[of sgn a] ..}
  \texttt{also have real-of-int (|a| * b) = of-nat (nat (abs a * b)) using b}}
  \texttt{by (metis abs-sgn mult-pos-pos mult-zero-left nat-int of-int-of-nat-eq of-nat-0 zero-less-imp-eq-int)}
  \texttt{also have .. = of-nat fact * (of-nat sq * of-nat sq) unfolding ab of-nat-mult}}
  \texttt{by simp}
\texttt{also have sqrt (of-int (sgn a) * (of-nat fact * (of-nat sq * of-nat sq))) =}
  \texttt{of-int (sgn a) * sqrt (of-nat fact) * of-nat sq}
\texttt{unfolding real-sqrt-mult by simp}
\texttt{finally have r: sqrt (real-of r) = real-of-int (sgn a) * real-of-int sq / real-of-int b * sqrt (real-of-int fact) by simp}
\texttt{let \texttt{?asqb = ma-of-rat (rat-of-int (sgn a) * rat-of-int sq / rat-of-int b)}}
\texttt{let \texttt{?f = ma-of-rat (rat-of-int fact)}}
\texttt{let \texttt{?sq = ma-sqrt ?f}}
\texttt{have sq: 0 \leq ma-rat \texttt{?f}} \texttt{ma-coeff \texttt{?f = 0 by ((transfer, simp)+)}}
\texttt{have compat: \texttt{\& m. (ma-compatible ?asqb m) = True}}
  \texttt{by (transfer, auto)}
\texttt{show sqrt (real-of r) = real-of

24
(let (a, b) = quotient-of (ma-rat r); (sq, fact) = prime-product-factor (nat (|a| + b));
    ma' = ma-of-rat (rat-of-int (sgn a) * rat-of-int sq / rat-of-int b)
in ma-times ma' (ma-sqrt (ma-of-rat (rat-of-nat fact))))

unfolding q ppf Let-def split
unfolding r
unfolding ma-times[symmetric, of ?asqb, unfolded compat if-True]
unfolding ma-sqrt-main[OF sq, symmetric]
unfolding ma-of-rat[symmetric]
by (simp add: of-rat-divide of-rat-mult)

qed

lemma mau-sqrt: sqrt (real-of-u r) = (if mau-coeff r = 0 then
    real-of-u (mau-sqrt r)
  else Code.abort (STR "cannot represent sqrt of irrational number") (λ -. sqrt (real-of-u r)))
using mau-sqrt-main[of r] by (cases mau-coeff r = 0, auto)

lemma mau-0: 0 = real-of-u (mau-of-rat 0) using ma-0 by (transfer, auto)

lemma mau-1: 1 = real-of-u (mau-of-rat 1) using ma-1 by (transfer, auto)

lemma mau-equal:
HOL.equal (real-of-u r1) (real-of-u r2) = mau-equal r1 r2 unfolding equal-real-def
using real-of-a-inj[of r1 r2]
by (transfer, transfer, auto)

lemma mau-ge-0: ge-0 (real-of-u x) = mau-ge-0 x using mau-ge-0
by (transfer, auto)

definition real-lt :: real ⇒ real ⇒ bool where real-lt = (<)

The following code equation terminates if it is started on two different inputs.

lemma real-lt [code equation]: real-lt x y = (let fx = floor x; fy = floor y in
(if fx < fy then True else if fx > fy then False else real-lt (x * 1024) (y * 1024)))
proof (cases floor x < floor y)
case True
  thus ?thesis by (auto simp: real-lt-def floor-less-cancel)
next
case False note nless = this
show ?thesis
proof (cases floor x > floor y)
case True
  from floor-less-cancel[of this] True nless show ?thesis
  by (simp add: real-lt-def)
next
case False
with nless show ?thesis unfolding real-lt-def by auto
For comparisons we first check for equality. Then, if the bases are compatible we can just compare the differences with 0. Otherwise, we start the recursive algorithm \texttt{real-lt} which works on arbitrary bases. In this way, we have an implementation of comparisons which can compare all representable numbers.

Note that in \texttt{Real-Impl} we did not use \texttt{real-lt} as there the code-equations for equality already require identical bases.

\textbf{lemma} \texttt{comparison-impl}:

\[
\text{real-of-u } x \leq \text{real-of-u } y \iff \text{real-of-u } x = \text{real-of-u } y \lor \\
(\text{if mau-compatible } x \text{ and } y \text{ then } \text{ge-0 } (\text{real-of-u } y - \text{real-of-u } x) \text{ else } \text{real-lt } (\text{real-of-u } x) (\text{real-of-u } y))
\]

\[
\text{real-of-u } x < \text{real-of-u } y \iff \text{real-of-u } x \neq \text{real-of-u } y \land \\
(\text{if mau-compatible } x \text{ and } y \text{ then } \text{ge-0 } (\text{real-of-u } y - \text{real-of-u } x) \text{ else } \text{real-lt } (\text{real-of-u } x) (\text{real-of-u } y))
\]

\textbf{unfolding} \texttt{ge-0-def} \texttt{real-lt-def} \texttt{by} (\texttt{auto simp del: real-of-u-inj})

\textbf{lemma} \texttt{mau-is-rat}:

\[
\text{is-rat } (\text{real-of-u } x) = \text{mau-is-rat } x
\]

\texttt{using} \texttt{ma-is-rat} \texttt{by} (\texttt{transfer}, \texttt{auto})

\textbf{lift-definition} \texttt{ma-show-real :: mini-alg \Rightarrow string is}

\[
\lambda (p,q,b). \text{let } sb = \text{shows } "\sqrt" \circ \text{shows } b \circ \text{shows } ";" ; \\
qb = (\text{if } q = 1 \text{ then } sb \text{ else if } q = -1 \text{ then } \text{shows } "-" \circ sb \text{ else } \text{shows } q \circ \\
\text{shows } "s" \circ sb) \text{ in} \\
(\text{if } q = 0 \text{ then } \text{shows } p \text{ else} \\
\text{if } p = 0 \text{ then } \text{qb} \text{ else} \\
\text{if } q < 0 \text{ then } ((\text{shows } p \circ \text{qb}) []) \\
\text{else } ((\text{shows } p \circ \text{shows } "+" \circ \text{qb}) [])).
\]

\textbf{lift-definition} \texttt{mau-show-real :: mini-alg-unique \Rightarrow string is} \texttt{ma-show-real}.

\textbf{overloading} \texttt{show-real \equiv show-real}

\textbf{begin}

\textbf{definition} \texttt{show-real}

\[
\text{where } \text{show-real } x \equiv \\
(\text{if } (\exists \ y. x = \text{real-of-u } y) \text{ then } \text{mau-show-real } (\text{THE } y. x = \text{real-of-u } y) \text{ else } [])
\]

\textbf{end}

\textbf{lemma} \texttt{mau-show-real}:

\[
\text{show-real } (\text{real-of-u } x) = \text{mau-show-real } x
\]

\textbf{unfolding} \texttt{show-real-def} \texttt{by simp}

\textbf{code-datatype} \texttt{real-of-u}

\textbf{declare} \texttt{[code drop:}

\[
\text{plus :: real } \Rightarrow \text{real } \Rightarrow \text{real} \\
\text{uminus :: real } \Rightarrow \text{real} \\
\text{times :: real } \Rightarrow \text{real } \Rightarrow \text{real}
\]

26
inverse :: real ⇒ real
floor :: real ⇒ int
sqrt
HOL.equal :: real ⇒ real ⇒ bool
ge-0
is-rat
less :: real ⇒ real ⇒ bool
less-eq :: real ⇒ real ⇒ bool
]]

lemmas mau-code-eqns [code] = mau-floor mau-0 mau-1 mau-uminus mau-inverse
mau-sqrt mau-plus mau-times mau-equal mau-ge-0 mau-is-rat
mau-show-real comparison-impl

Some tests with small numbers. To work on larger number, one should additionally import the theories for efficient calculation on numbers

value [101.1 * (sqrt 18 + 6 * sqrt 0.5)]
value [324 * sqrt 7 + 0.001]
value 101.1 * (sqrt 18 + 6 * sqrt 0.5) = 324 * sqrt 7 + 0.001
value 101.1 * (sqrt 18 + 6 * sqrt 0.5) > 324 * sqrt 7 + 0.001
value show (101.1 * (sqrt 18 + 6 * sqrt 0.5))
value (sqrt 0.1 ∈ Q, sqrt (− 0.09) ∈ Q)

end

Acknowledgements

We thank Bertram Felgenhauer for interesting discussions and especially for mentioning Cauchy’s mean theorem during the formalization of the algorithms for computing roots.

References


